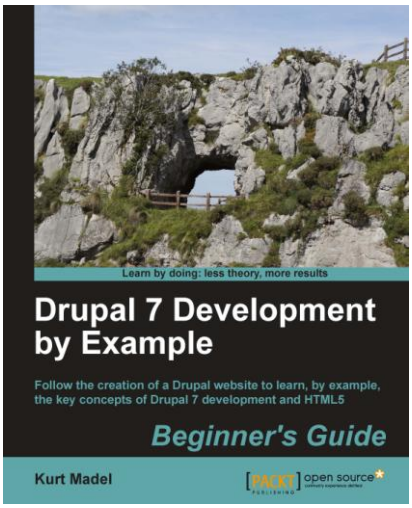
[PACKT] open source
community experience distilled
PUBLISHING

Drupal 7 Development by Example Beginner's Guide

Kurt Madel



Chapter No. 3
"HTML5 Integration for Drupal 7 and More Module Development"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.3 "HTML5 Integration for Drupal 7 and More Module Development"

A synopsis of the book's content

Information on where to buy this book

About the Author

Kurt Madel is a Senior Manager and developer for Captech Consulting in Richmond, VA. He has worked on open source CMS projects for over six years. Kurt contributes regularly to `Drupal.org`, and is the maintainer of several modules. In addition to Drupal, Kurt has been doing Java EE development since 2000, and has focused on mobile web development over the last two years. When he is not writing or programming, Kurt enjoys cycling and spending time with his wife and four boys.

Drupal 7 Development by Example Beginner's Guide

This book is a hands-on, example-driven guide to programming Drupal websites. Discover a number of new features for Drupal 7 through practical and interesting examples while building a fully functional recipe sharing website. Learn about web content management, multi-media integration, and new features for developers in Drupal 7.

With this book you will:

- Learn to build cutting edge websites with Drupal 7
- Discover important concepts for HTML5 and why it's time to start building websites with HTML5, if you haven't already
- Learn the important patterns for JavaScript and AJAX in Drupal 7
- Realize interesting ways to integrate multi -media with Drupal 7
- Find out how becoming more involved with the Drupal development community can help you build better websites
- Set up a development environment, and learn to use Git and Drush
- Uncover how much fun it can be to build websites with Drupal 7

What This Book Covers

Chapter 1, Getting Set up, walks through setting up a Drupal development environment before diving into Drupal development.

Chapter 2, Custom Content Types and an Introduction to Module Development, will explain how to configure Drupal content types and begin Drupal development with an introduction to creating custom Drupal modules.

Chapter 3, HTML5 Integration for Drupal 7 and More Module Development, will continue with some Drupal development examples, and show how we can integrate HTML5 with Drupal 7.

Chapter 4, Introduction to Drupal 7 Theme Development, will explain about custom theme development and Drupal 7 render arrays.

Chapter 5, Enhancing the Content Author's User Experience, will configure WYSIWYG for Drupal 7 and show how we can improve the content author user experience with some custom development examples.

Chapter 6, Adding Media to our Site, will explain how the Drupal 7 Media module makes multi -media integration for Drupal better than ever, keeping in mind that a site without any multi media is just text.

Chapter 7, How Does it Taste – Getting Feedback, will show some ways to provide visitor site interaction in Drupal as one way to extend the amount of time a visitor spends on your site is to provide ways for them to interact with it.

Chapter 8, Recipe Lists and More with Views, will show some of the more advanced features of the Views module, and find out why Views is the most popular contrib module on drupal.org.

Chapter 9, Rotating Banners and Project Promotion, will show how to use Views to display images in a compelling way, because the Views module offers a lot more than just displaying some fields.

Chapter 10, Test Your Code with SimpleTest, will test the custom code that you have written.

Chapter 11, Introduction to the Features Module and Configuration Management, will show how the Features module can help us manage the code that you have written for the Drupal site that you created.

3

HTML5 Integration for Drupal 7 and More Module Development

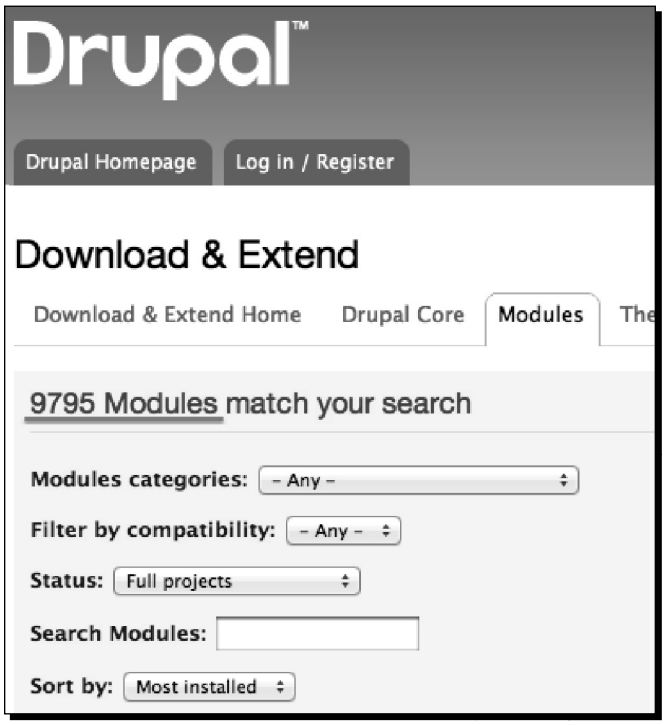
HTML5 *is not exactly new on the technology scene. If you have done any mobile web development, then you will certainly know how prevalent HTML5 is becoming. Even though there is an active initiative around HTM5 for Drupal 8 (*`http://drupal.org/community-initiatives/drupal-core/html5`*), the adoption of HTML5 was not far enough along to be included as part of Drupal 7 core.*

HTML5 is one of the main ingredients of many of the upcoming development examples in this book.

In this chapter, we will explore some of the different modules and options for using HTML5 with Drupal 7, and we will enhance our custom-developed `d7dev` module to include certain HTML5 features.

Modules are a key element of what makes Drupal so attractive for both developers and non-developers. The multitude of available contributed modules makes Drupal attractive to non-developers looking for certain features, while the ease of module development and integration of such modules, with Drupal core, and other contributed modules, is what makes Drupal popular with developers.

Evidence of this popularity is displayed by the availability of almost 10,000 contributed modules on `http://drupal.org/project/modules/`. Therefore, in addition to updating our existing module with the HTML5 features, we will continue the development emphasis on module development. We are going to develop a new compound field module.



This chapter, with its HTML5-driven code examples, will serve as a foundation for many of the code examples in the following chapters.

First things first—changing our DOCTYPE

The `DOCTYPE` of our content may seem like an odd thing to be mentioning at this point. However, since HTML5 will be a major concept and building block for the rest of this book, it is important to get off to a good start. This all starts with the HTML5 `DOCTYPE`.

Drupal 7 defaults to an XHTML `DOCTYPE`:

```
<!DOCTYPE html PUBLIC "-//W3C//DTD
XHTML+RDFa 1.0//EN"   "http://www.w3.org/
MarkUp/DTD/xhtml-rdfa-1.dtd">
```

Whereas the HTML5 DOCTYPE is much simpler:

```
<!DOCTYPE html>
```

The HTML5 DOCTYPE is fully supported by all modern browsers, so at this point, there is not a good reason for not using it.

The `DOCTYPE` is hardcoded in the `html.tpl.php` file, and can be overridden in any theme, by creating your own version of that template file. However, we aren't currently using a custom theme (that's coming in the next chapter), and we don't want to modify any of the core themes (see the following information box for an explanation of why you shouldn't modify or hack core). So, there is a simple solution: the contrib HTML5 Tools module.

Do not hack core. This is a phrase that you will come across time and time again, as you do more and more custom Drupal development. The basic idea behind the phrase is that Drupal provides so many ways to modify the behavior of a Drupal site without modifying any core modules, themes or other files, that you will only cause yourself unnecessary grief in regards to upgrading core, and dealing with possible core issues if you start modifying core. An in-depth explanation of why it is a bad practice to hack core can be found at `http://drupal.org/best-practices/do-not-hack-core`.

Time for action – installing the HTML5 Tools module

The HTML5 Tools module (`http://drupal.org/projects/html5_tools`) will provide an HTML5-compliant `DOCTYPE` and will provide support for other HTML5 features.

Once again, we will use Drush to download and enable the module.

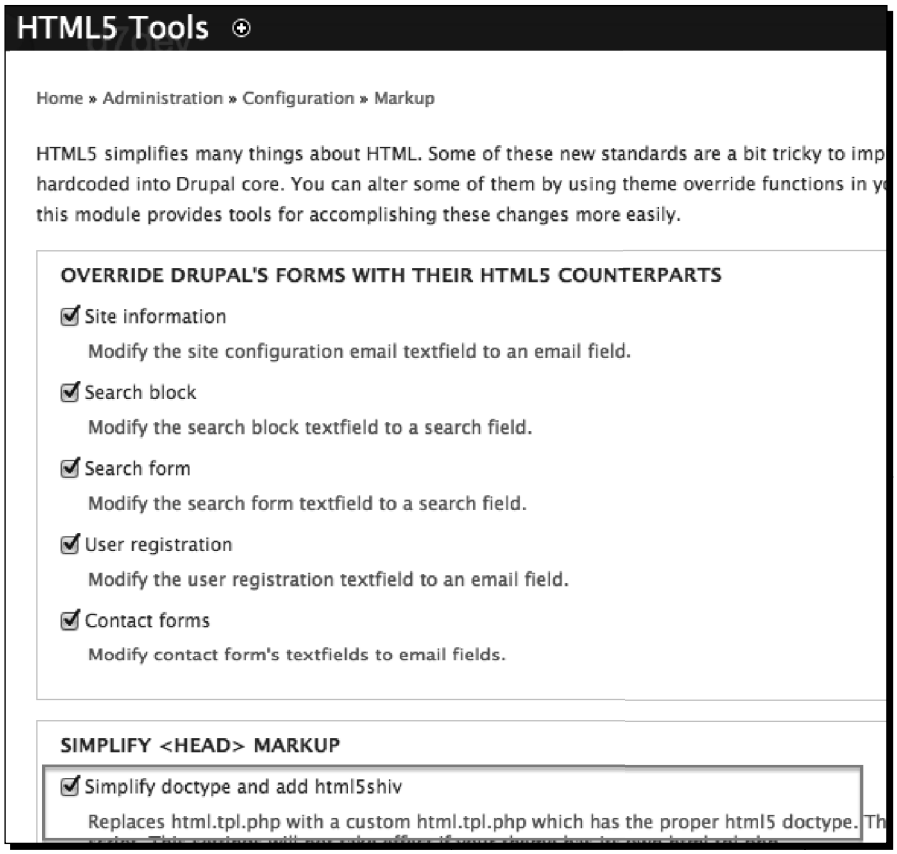
```
C:\xampp\htdocs\d7dev>drush dl html5_tools
Project html5_tools (7.x-1.1) downloaded to                [success]
C:/xampp/htdocs/d7dev/sites/all/modules/html5_tools.

C:\xampp\htdocs\d7dev>drush en html5_tools
The following projects have unmet dependencies:
html5_tools requires elements
Would you like to download them? (y/n): y
Project elements (7.x-1.2) downloaded to                   [success]
C:/xampp/htdocs/d7dev/sites/all/modules/elements.
The following extensions will be enabled: html5_tools, elements
Do you really want to continue? (y/n): y
elements was enabled successfully.                         [ok]
html5_tools was enabled successfully.                      [ok]
```

In addition to providing an HTML5-compliant `DOCTYPE` (that is enabled, by default, just by enabling the module), the HTML5 Tools module, and its dependent Elements module (`http://drupal.org/projects/elements`), a number of other HTML5 features provided by the HTML5 Tools module include the following:

- Overrides Drupal core forms with HTML5 counterparts
- Simplifies the `head` markup for HTML5 specified `style`, `javascript`, and `meta` tags
- Uses the new HTML5 `time` element for content and comments publication dates

For a more comprehensive listing and explanation of what the HTML5 Tools modules does, take a look at the administrative configuration page for **HTML5 Tools** by selecting **Configuration** from the **Admin** toolbar, and selecting the **HTML5 Tools** link from the **Markup** section.



HTML5, RDFa, and Microdata

As we discussed in the previous chapter, one aspect of HTML5 is semantic markup, or the ability to describe your content in a meaningful way with a vocabulary, and apply the vocabulary to the markup of your content. Drupal 7 has such a semantic capability baked right into core, and it is called **RDFa**. However, the way that RDFa is integrated with Drupal 7 is not HTML5-compliant. Furthermore, although RDFa will be supported by HTML5, the HTML5 Microdata specification (`http://dev.w3.org/html5/md/`) was specifically designed with HTML5 in mind, and there is a Microdata module for Drupal 7 (`http://drupal.org/project/microdata`).

So, let us enhance our `d7dev` Recipe content type with some HTML5 Microdata that will make it semantically identified as an `http://schema.org/Recipe` item.

This section is not intended to start a flame war, and is not trying to make the point that Microdata is better than RDFa. Or, that Drupal 7 made a big mistake by including the XHTML flavor of RDFa instead of HTML5 RDFa or Microdata. The HTML5 version of RDFa, and for that matter Microdata, didn't even exist when the new feature set for Drupal 7 was frozen. The point of this section is that things change in the web world between major Drupal releases, and sometimes those changes need to be addressed sooner rather than later. This is the real point of this section: *showing how easy it is to adapt Drupal to the latest and the greatest new web standards*. The Microdata schema that we are going to associate with our Recipe content type is part of the schema.org project that is backed by Google, Yahoo, and Microsoft. If you would like to learn more about Microdata, you can find an excellent introduction to understanding and using Microdata at: `http://diveintohtml5.org/extensibility.html`.

Time for action – installing the Microdata module

Once again, we are going to use Drush to download and install the Microdata module. Start by opening a Command Prompt for Windows or Terminal for Mac, change to your `d7dev` working folder, type the following commands, and you should see the following responses:

```
C:\xampp\htdocs\d7dev>drush dl microdata
There is no recommended release for project microdata.
Choose one of the available releases:
 [0]  :  Cancel
 [1]  :  7.x-1.x-dev      -  2012-Feb-24  -  Development
 [2]  :  7.x-1.0-alpha4   -  2012-Feb-22  -  Supported

2
```

```
Project microdata (7.x-1.0-alpha4) downloaded to          [success]
C:/xampp/htdocs/d7dev/sites/all/modules/microdata.

C:\xampp\htdocs\d7dev>drush en microdata
The following projects have unmet dependencies:
microdata requires entity
Would you like to download them? (y/n): y
Project entity (7.x-1.0-rc1) downloaded to          [success]
C:/xampp/htdocs/d7dev/sites/all/modules/entity.
The following extensions will be enabled: microdata, entity
Do you really want to continue? (y/n): y
entity was enabled successfully.                       [ok]
microdata was enabled successfully.                    [ok]
```

Drush will actually modify the database of your Drupal installation when running certain commands, such as `drush en`. In order for Drush to modify the correct database, you must run the `Drush` command within the `root` Drupal `install` folder of your Drupal site. Otherwise, you would have to include which specific Drupal instance you would like Drush to modify with the `-r` argument: `drush -r /Applications/MAMP/htdocs/d7dev`

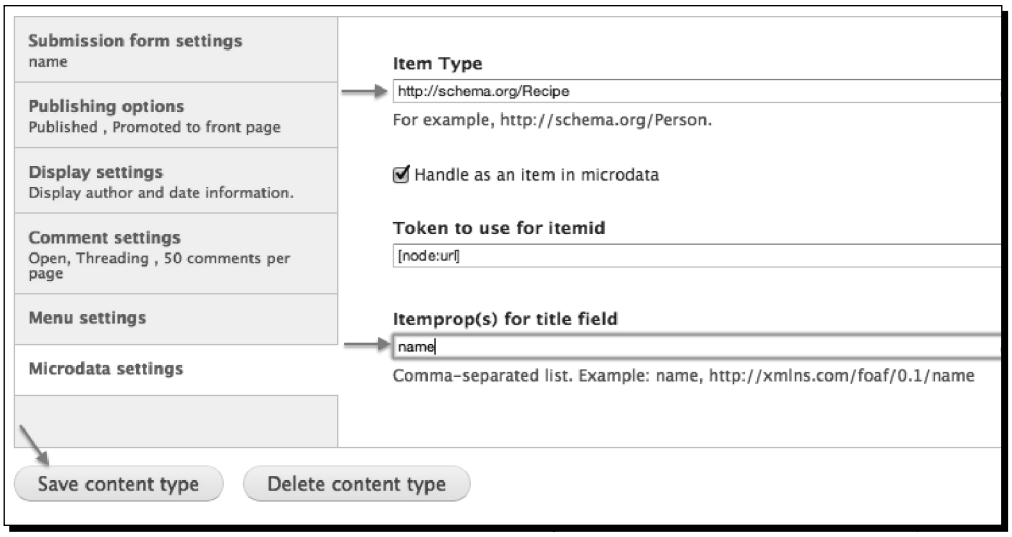
What just happened?

We have now installed the Microdata module and the Entity module as its dependency. Now that we have installed the Microdata module, let us put it to use, before we move on to some more development examples. We will configure our Recipe content type and its fields to utilize the functionality of the Microdata module.

Time for action – configuring Microdata for our Recipe content type

1. Navigate to the **Structure | Content Types** page, and select the **edit** link for our Recipe content type.

2. On the **Recipe content type edit** page, select the **Microdata settings** tab, and you will see a screen similar to the following, only without values in the **Field property(s)** input:



3. Type `http://schema.org/Recipe` as the value for the **Item Type** field, and `name` as the value for the **Itemprop(s) for title field** field. Click on the **Save content type** button.

4. Now, click on the **Home** button at the far left of the **Admin** toolbar, and select any of the Recipe content items from our Recipe List block on the right side of the page.

5. Next, right-click anywhere on the recipe item page, and select **View source (IE)** or **View Page Source** from your browser's menu.

6. In the source view of our page, scroll down to approximately *line 182*, and look for the following markup. You will see that the Microdata `itemscope` and `itemtype` attributes have been set on the enclosing `DIV` of our recipe content node:

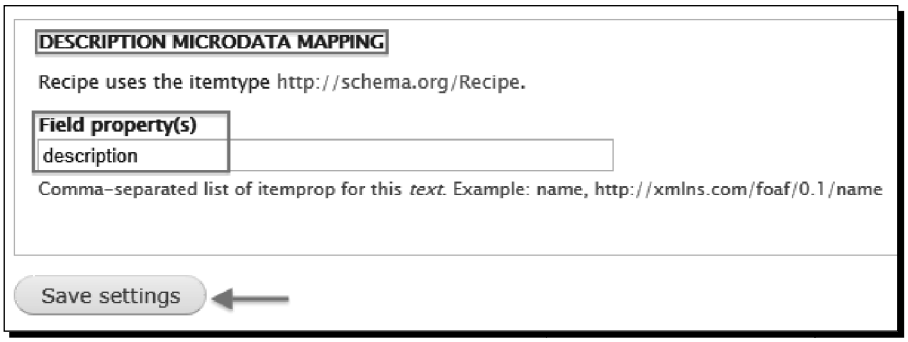
```
<div id="node-51" class="node node-recipe node-promoted node-full
  clearfix" itemid="/d7dev/node/51" itemscope=""
  itemtype="http://schema.org/Recipe">.
```

7. However, if you inspect the HTML source any further, you will notice that none of the fields of our Recipe content node have any Microdata attributes applied to them. For example, the `DIV` wrapper for our `description` field will look as follows:

```
<div class="field field-name-field-description field-type-text-
long field-label-above">
```

8. As previously mentioned in this chapter, the Microdata module supports the ability to specify the `itemProp` values for any text field type. So, from the **Admin** toolbar, select **Structure | Content types**, and select the **manage fields** link for our Recipe content type.

9. Click on the **edit** link for the **description** field, and scroll down to the **Description Microdata Mapping** section on the **description field edit** page. Referring to the `http://schema.org/Recipe` definition, we will set the **Field property(s)** input to **description**, as shown in the following screenshot:



The `node id` will most likely not be exactly the same in your environment, so look for a `DIV` element with the class as specified previously.

10. Click on the **Save settings** button to save the updated configuration for the **description** field.

11. Now, we will repeat the same steps for the rest of the text fields for our Recipe content type, setting **Field property of each** under the **Description Microdata Mapping** section accordingly:

ingredients	ingredients
recipeInstructions	recipeInstructions
recipeYield	recipeYield

12. Now, to test the output of the Microdata module, we will once again click on the **Home** button at the far left of the **Admin** toolbar, and select any of the recipe content items from our Recipe List block on the right side of the page.

13. Next, right-click anywhere on the **recipe item** page, and select **View source (IE)** or **View Page Source** from your browser's menu.

14. In the source view of our page, scroll down to approximately *line 191*, and look for the following markup:

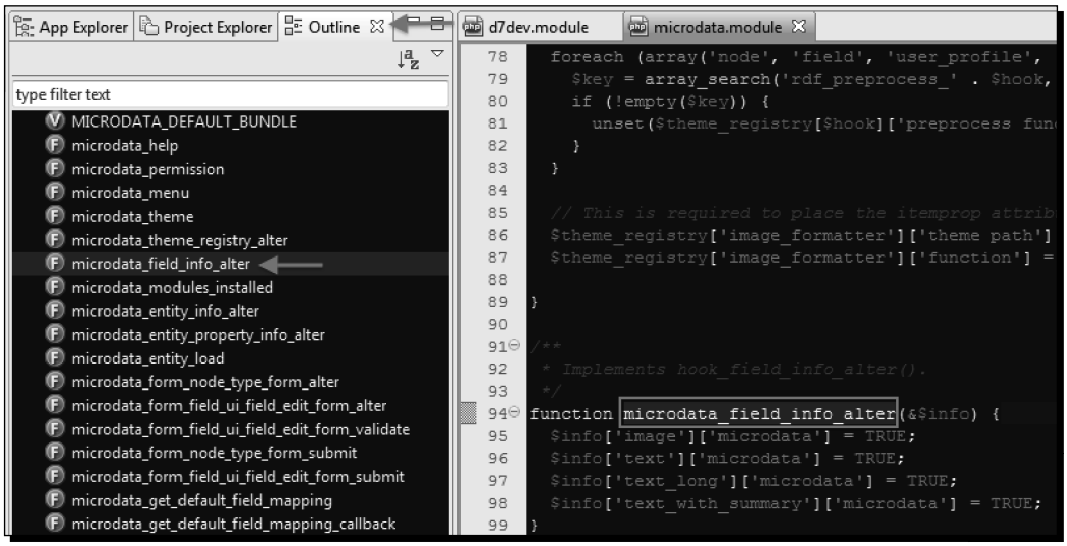
```
<div class="field field-name-field-description
  field-type-text-long field-label-above">
  <div class="field-label">
description: 
</div>
<div class="field-items">
  <div class="field-item even" itemprop="description">
```

What just happened?

We enhanced our Recipe content type with Microdata metadata.

Earlier in its development, the Microdata module did not support the Drupal core `number_integer` field type, the field type that we used for the `prepTime` and `cookTime` fields of our Recipe content type. In order to provide the ability to enable the Microdata field properties for the `cookTime` and `prepTime` integer fields on our Recipe content type, we just need to implement the `hook_field_info_alter` hook, just as the Microdata module did for image and text fields.

The latest release of the Microdata module at the time of the publishing of this book, `7.x-1.0-alpha4`, actually included support for integer fields. Therefore, you will not need to add the following code to your custom `d7dev` module; rather, this code example illustrates the power of the Drupal community at work, by showing how local changes can eventually make their way back into core and the contributed module code.



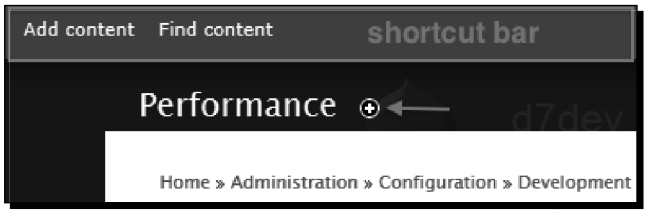
Take a close look at this screenshot, and you will see that at one point, the Microdata module only supports field mappings for the image field and text fields. So, in order to add support for the `number_integer` field, all we would have to do is implement the `hook_field_info_alter` hook just as the Microdata module. But for the `number_integer` field, the code would look something as follows:

```
/**
 * Implements hook_field_info_alter().
 */
function d7dev_field_info_alter(&$info) {
  $info['number_integer']['microdata'] = TRUE;
}
```

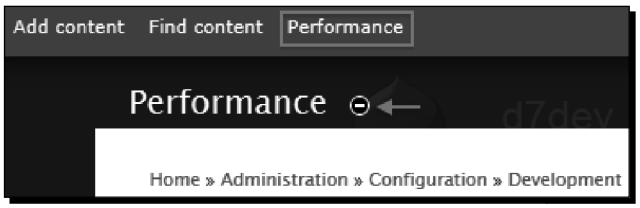
That is all it would take to add the Microdata support for core integer fields.

Drupal Shortcut Bar

There are many Drupal core hooks that require you to clear the Drupal cache in order for them to take effect. The `hook_field_info_alter` hook is such a hook, and it can get somewhat tiresome navigating to the **Performance administrative** page over and over again, when testing some new code. Luckily, Drupal 7 added the administrative UX component called the **shortcut bar**. The shortcut bar allows you to create a shortcut link to any administrative page. So, we can use this functionality to add the **Performance settings** page to our shortcut bar. Click on the plus icon next to the **Performance** heading, and it will show up as a link in your shortcut bar.



And notice after you add it, there is a minus icon next to the Performance heading. Therefore, you can easily remove the shortcut if you decide you don't use it much anymore, or your shortcut bar is getting crowded.



Anytime that you find yourself going to a certain administrative configuration page over and over, you should add it to your shortcut bar.

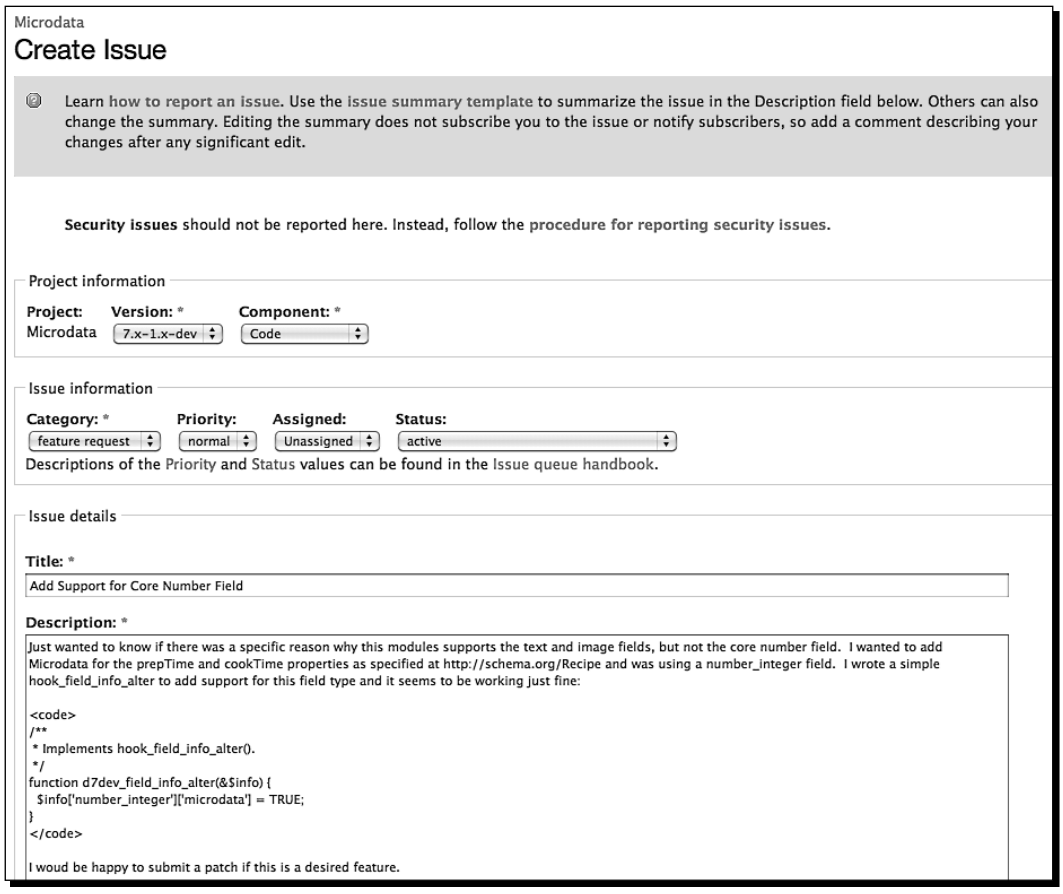
Drupal development and the Drupal community

As discussed in *Chapter 1*, *Getting Set up*, being a good Drupal developer means being aware of and active in the Drupal community. We have an opportunity to share this small bit of code we wrote with the Drupal community by adding an issue to the Microdata project issue queue (`http://drupal.org/project/issues/microdata`), suggesting that the Microdata module should add support for the core number field. This will not only help other Drupal users who would like this capability, it will also be custom code that we will no longer have to support by ourselves if it gets added to the Microdata module.

Time for action – creating issues in Contrib modules' issue queues

I am going to follow my own advice, and walk you through the process of adding an issue to a Drupal module's issue queue.

1. Open a browser, log into `http://drupal.org`, and navigate to `http://drupal.org/project/issues/microdata`.
2. Click on the **Create new issue** link under the **Issues for Microdata** heading.
3. Next, fill out all of the fields for **Create Issue form**. Here is a screenshot of the issue that I submitted for adding support for number field types:



4. After you are done filling out the form, click on the **Save** button at the bottom of the page.

What just happened?

We created an issue in the Microdata module issue queue, and if we follow up with the issue at `http://drupal.org/node/1291634`, we will see that the module maintainer has actually implemented our feature request issue.

Issue Summary

Just wanted to know if there was a specific reason why this modules supports the text and image
add Microdata for the prepTime and cookTime properties as specified at http://schema.org/Recip
simple hook_field_info_alter to add support for this field type and it seems to be working just fine

```
/**
 * Implements hook_field_info_alter().
 */
function d7dev_field_info_alter(&$info) {
  $info['number_integer']['microdata'] = TRUE;
}
```

I woud be happy to submit a patch if this is a desired feature.

Login or register to post comments

Comments

#1 Posted by linclark on *September 26, 2011 at 11:20pm*

That would be great. IIRC, the only reason I hadn't added it yet is because I haven't written t
number fields to the SimpleTest for core fields, then this would be ready to go. I'm not sure
need it.

#2 Posted by linclark on *December 21, 2011 at 3:02pm*

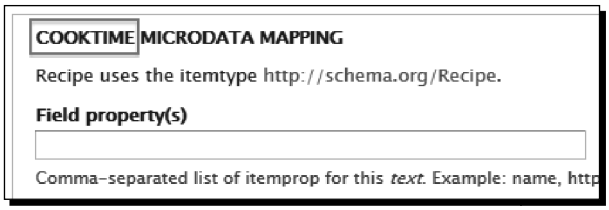
Status: active » fixed

This is fixed with commit http://drupalcode.org/project/microdata.git/commit/414cdea.

Time for action – adding Microdata mappings for Recipe number_integer fields

Now that the Microdata supports the `number_integer` fields, we will update all of our Recipe content type `number_integer` fields with a Microdata mapping.

1. Now, go back to the **field settings** page for the `cookTime` field, and you will see that we now have the **Field property(s)** input. So, go ahead and enter `cookTime` in that field.



2. Now, once again we will navigate to a Recipe content item page, and view the HTML source. The `cookTime` field will look as follows:

```
<div class="field field-name-field-cooktime field-type-number-
  integer field-label-above">
  <div class="field-label">cookTime: </div>
  <div class="field-items">
    <div class="field-item even" itemprop="cookTime">
      84 and
      <sup>14</sup>&frasl;
      <sub>15</sub> hours
    </div>
  </div>
</div>
```

3. Finally, go back to the Recipe content type manage fields screen, and repeat the previous steps for the `prepTime` field.

What just happened?

We enabled the Microdata support for the `number_integer` fields of our Recipe content type.

Now, all of the fields of our current Recipe content type fields are associated with Microdata properties. With the help of the contributed Microdata module, adding Microdata support for our `cookTime` and `prepTime` Recipe fields was a straightforward task.

NutritionInformation module

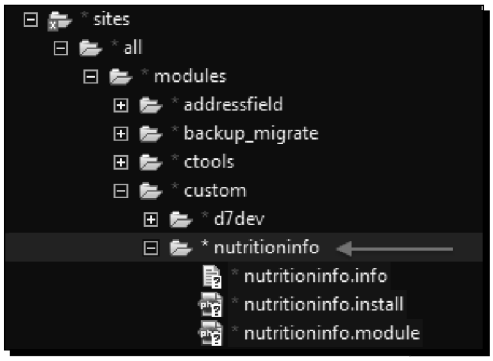
One of the `http://schema.org/Recipe` properties that we did not include with our Drupal Recipe content type is the `NutritionInformation` property. The reason for that is because the `NutritionInformation` property is itself an `itemType` from `http://schema.org`, and as such is made up of a number of its own individual properties. In order to add `NutritionInformation` to our custom Recipe content type, we are going to need to create a custom Drupal compound field module that is based on the specification at `http://schema.org/NutritionInformation`.

Time for action – developing a custom module for a compound NutritionInformation field

Rather than adding the code to create this compound field to our existing module, we are going to create a new module, as it is possible that it is something that could be useful to the Drupal community as a whole, and we may want to eventually contribute it to drupal.org.

1. In **Aptana Studio**, create a new folder named `nutritioninfo` in the `/sites/all/modules/custom` directory.

2. Create the `.module`, `.info`, and `.install` files with the same name as the folder - `nutritioninfo`, and you should have a folder that looks similar to the following screenshot:



3. Now, open the `nutrtioninfo.info` file and add the following configuration:

```
name = Nutrition Information Field
description = Defines a nutrition information field type based on
the Microdata spec at http://schema.org/NutritionInformation
core = 7.x
package = Fields
```

4. I always find that development is easier when you are able to look at an example. So, we are going to use Drush to download the contributed **Address Field module** (`http://drupal.org/project/addressfield`) to use as an example for our own compound field module:

The Address Field module is a good example of a compound field module. So, the code from that module will provide some good examples for implementing our own compound field module.

```
C:\xampp\htdocs\d7dev>drush dl addressfield-7.x-1.0-beta2
Project addressfield (7.x-1.0-beta2) downloaded to [success]
C:/xampp/htdocs/d7dev/sites/all/modules/addressfield.
Project addressfield contains 2 modules: addressfield_example,
addressfield.
```

5. So, now that we have downloaded the Address Field module, let's take a look at its code. In **Aptana Studio**, open the `addressfield.module`, and `addressfield.install` files located at `/sites/all/modules/addressfield`.

6. Next, we are going to use the following code from the `addressfield.install` file as a starting point for our `nutritioninfo.install` file.

```
<?php

/**
 * Implements hook_field_schema()
 */
function addressfield_field_schema() {
  $columns = array(
    'country' => array(
      'description' => 'Two letter ISO country code of this
address.',
      'type' => 'varchar',
      'length' => 2,
      'not null' => FALSE,
      'default' => '',
    ),
```

7. Now, we will rename the function to `nutritioninfo_field_schema`, so that our `hook_field_schema` code looks as follows:

```
<?php

/**
 * Implements hook_field_schema()
 */
function nutritioninfo_field_schema() {
  $columns = array(
    'country' => array(
      'description' => 'Two letter ISO country code of this
address.',
      'type' => 'varchar',
      'length' => 2,
```

```
        'not null' => FALSE,
        'default' => '',
      ),
    );

    return array(
      'columns' => $columns,
    );
}
```

8. The `hook_field_schema` hook (`http://api.drupal.org/api/drupal/modules--field--field.api.php/function/hook_field_schema/7`) allows us to define a database schema for storing our custom field information, and is automatically detected by Drupal, as long as it is in the `.install` file of our module. Now, we need to replace the `country` column as specified by the code we copied from the `addressfield_field_schema` function and add columns for the rest of the properties defined at `http://schema.org/NutritionInformation`.

9. After adding the column specifications for all of the `NutritionInformation` properties, our `nutritioninfo_field_schema` function will look as follows:

```
/**
 * Implements hook_field_schema()
 */
function nutritioninfo_field_schema() {
  $columns = array(
    'calories' => array(
      'description' => 'The number of calories.',
      'type' => 'varchar',
      'length' => 255,
      'not null' => FALSE,
      'default' => '',
    ),
    'carbohydrate_content' => array(
      'description' => 'The number of grams of carbohydrates.',
      'type' => 'varchar',
      'length' => 255,
      'not null' => FALSE,
      'default' => '',
    ),
    'cholesterol_content' => array(
      'description' => 'The number of milligrams of cholesterol.',
      'type' => 'varchar',
```

```
        'length' => 255,
        'not null' => FALSE,
        'default' => '',
      ),
      'fat_content' => array(
        'description' => 'The number of grams of fat.',
        'type' => 'varchar',
        'length' => 255,
        'not null' => FALSE,
        'default' => '',
      ),
      'fiber_content' => array(
        'description' => 'The number of grams of fiber.',
        'type' => 'varchar',
        'length' => 255,
        'not null' => FALSE,
        'default' => '',
      ),
      'protein_content' => array(
        'description' => 'The number of grams of protein.',
        'type' => 'varchar',
        'length' => 255,
        'not null' => FALSE,
        'default' => '',
      ),
      'saturated_fat_content' => array(
        'description' => 'The number of grams of saturated fat.',
        'type' => 'varchar',
        'length' => 255,
        'not null' => FALSE,
        'default' => '',
      ),
      'serving_size' => array(
        'description' => 'The serving size, in terms of the number
of volume or mass.',
        'type' => 'varchar',
        'length' => 255,
        'not null' => FALSE,
        'default' => '',
      ),
      'sodium_content' => array(
        'description' => 'The number of milligrams of sodium.',
```

```
        'type' => 'varchar',
        'length' => 255,
        'not null' => FALSE,
        'default' => '',
      ),
      'sugar_content' => array(
        'description' => 'The number of grams of sugar.',
        'type' => 'varchar',
        'length' => 255,
        'not null' => FALSE,
        'default' => '',
      ),
      'trans_fat_content' => array(
        'description' => 'The number of grams of trans fat.',
        'type' => 'varchar',
        'length' => 255,
        'not null' => FALSE,
        'default' => '',
      ),
      'unsaturated_fat_content' => array(
        'description' => 'The number of grams of unsaturated fat.',
        'type' => 'varchar',
        'length' => 255,
        'not null' => FALSE,
        'default' => '',
      ),
    );

    return array(
      'columns' => $columns,
    );
  }
```

We reformatted the `NutritionInformation` schema property names to be all lower case instead of camel case, as lowercase with underscores is in line with Drupal coding standards. An overview of Drupal coding standards is available at `http://drupal.org/coding-standards`, and we will take a more in-depth look at Drupal coding standards in *Chapter 8, Recipe Lists and More with Views*.

10. Now we are going to move onto the nutritioninfo.module file and will open the addressfield.module to look at it implementation of hook_field_info. And once again, we will copy the contents of the addressfield function and paste it into our nutritioninfo.module file and modify it so that it looks like the following:

```
/**
 * Implements hook_field_info()
 */
function nutritioninfo_field_info() {
  $fields = array();

  $fields['nutritioninfo'] = array(
    'label' => t('Nutrition Information'),
    'description' => t('A field type used for storing nutrition
information as defined by the Microdata spec at http://schema.org/
NutritionInformation.'),
    'settings' => array(),
    'instance_settings' => array(),
    'default_widget' => 'nutritioninfo_standard',
    'default_formatter' => 'nutritioninfo_default',  );

  return $fields;
}
```

11. Now, we are going to implement two more hooks that are required by Drupal 7, when defining a custom field – `hook_field_validate` and `hook_field_is_empty`:

```
/**
 * Implements hook_field_validate().
 */
function nutritioninfo_field_validate($entity_type, $entity,
$field, $instance, $langcode, $items, &$errors) {
  //at this point we will not validate anything, but will revisit
}

/**
 * Implements hook_field_is_empty().
 */
function nutritioninfo_field_is_empty($item, $field) {
  //the nutrition field is empty if all of its properties are
empty
  return empty($item['calories'])
    && empty($item['carbohydrate_content'])
```

```
    && empty($item['cholesterol_content'])
    && empty($item['fat_content'])
    && empty($item['fiber_content'])
    && empty($item['protein_content'])
    && empty($item['saturated_fat_content'])
    && empty($item['serving_size'])
    && empty($item['sodium_content'])
    && empty($item['sugar_content'])
    && empty($item['trans_fat_content'])
    && empty($item['unsaturated_fat_content']);
}
```

12. Next, we need to tell Drupal how to handle our compound field on the node edit form. We will add `hook_field_widget_info` to make Drupal aware of our custom widget, and then `hook_field_widget_form` to actually add the form components to the node form:

```
/**
 * Implements hook_field_widget_info()
 */
function nutritioninfo_field_widget_info() {
  $widgets = array();

  $widgets['nutritioninfo_standard'] = array(
    'label' => t('Nutrition Information form'),
    'field types' => array('nutritioninfo'),    ),
  );

  return $widgets;
}

/**
 * Implements hook_field_widget_form()
 */
function nutritioninfo_field_widget_form(&$form, &$form_state,
$field, $instance, $langcode, $items, $delta, $element) {
  $settings = $form_state['field'][$instance['field_name']]
[$langcode]['field']['settings'];

  $fields = array(
    'calories' => t('Calories'),
    'carbohydrate_content' => t('Carbohydrate Content'),
    'cholesterol_content' => t('Cholesterol Content'),
```

```
    'fat_content' => t('Fat Content'),
    'fiber_content' => t('Fiber Content'),
    'protein_content' => t('Protein Content'),
    'saturated_fat_content' => t('Saturated Fat Content'),
    'serving_size' => t('Serving Size'),
    'sodium_content' => t('Sodium Content'),
    'sugar_content' => t('Sugar Content'),
    'trans_fat_content' => t('Trans Fat Content'),
    'unsaturated_fat_content' => t('Unsaturated Fat Content'),
  );

  foreach ($fields as $key => $label) {
    $value = isset($items[$delta][$key]) ? $items[$delta][$key] :
'';
    $element[$key] = array(
      '#attributes' => array('class' => array('edit-nutrition-
field'), 'title' => t('')),
      '#type' => 'textfield',
      '#size' => 3,
      '#maxlength' => 3,
      '#title' => $label,
      '#default_value' => $value,
      '#prefix' => '&lt;div class="nutrition-field nutrition-' .
$key . '-field">',
      '#suffix' => '&lt;/div>',
    );
  }
  return $element;
}
```

13. Now, we need to add some hooks for formatting our compound field, when displaying a content item. We will need to add two more hooks for that: `hook_field_formatter_info` and `hook_field_formatter_view`:

```
/**
 * Implements hook_field_formatter_info()
 */
function nutritioninfo_field_formatter_info() {
  return array(
    'nutritioninfo_default' => array(
      'label' => t('Default'),
      'field types' => array('nutritioninfo'),
    ),
```

```
  );
}

/**
 * Implements hook_field_formatter_view().
 */
function nutritioninfo_field_formatter_view($entity_type, $entity,
$field, $instance, $langcode, $items, $display) {
  $element = array();

  switch ($display['type']) {
    case 'nutritioninfo_default':
      $headers = array(
        t('Calories'),
        t('Carbohydrate Content'),
        t('Cholesterol Content'),
        t('Fat Content'),
        t('Fiber Content'),
        t('Protein Content'),
        t('Saturated Fat Content'),
        t('Serving Size'),
        t('Sodium Content'),
        t('Sugar Content'),
        t('Trans Fat Content'),
        t('Unsaturated Fat Content'),
      );

  $element[0]['#markup'] = theme('table', array('header' =>
$headers, 'rows' => $items));
  break;
  }
  return $element;
}
```

14. All right, now it is time to enable our new module. Open up our `d7dev` Drupal site in your favorite browser, and click on the **Modules** link in the **Admin** toolbar. Scroll down to the **Fields** section, and check the checkbox next to our new **Nutrition Info Field** module.

15. Finally, scroll to the bottom of the page, and click on the **Save configuration** button.

What just happened?

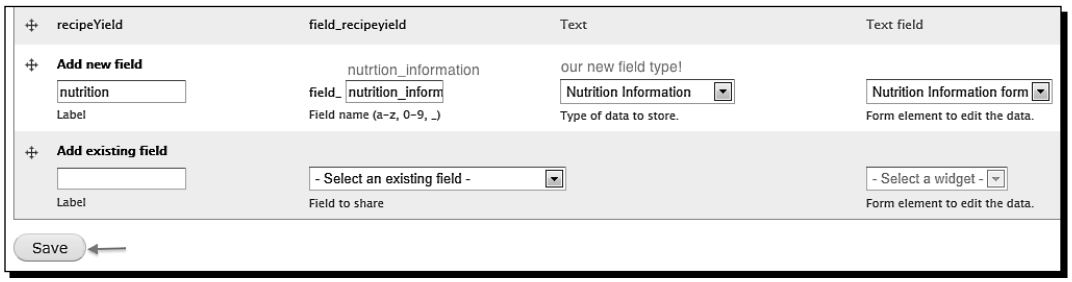
That was some serious development. We created a fairly complex custom module, and now have a field that offers a more complete Recipe content type.

Time for action – updating the Recipe content type to use the NutritionInformation field

Now, let's put our new module to use and add our new compound field to our Recipe content type, using our new custom `nutritioninfo` field for the `NutrionInformation` property of the `http://schema.org/Recipe` definition.

1. Go to the **Manage Fields** configuration page for the Recipe content type: `http://localhost/d7dev/#overlay=admin/structure/types/manage/recipe/fields`.

2. Now, add a new field with the following settings—label: `nutrition`, name: `field_nutrition_information`, type: `Nutrition Information`, widget: `Nutrition Information form` (the default), and click on the **Save** button.



3. There is nothing to set for **Field Settings**, so just click on the **Save field settings** button. On the **Recipe Settings** screen, enter `Nutrition information about the recipe` for the **Help text**, and then click on the **Save** button at the bottom of the screen.

4. Next, click on the **Find content** link in the **Shortcuts** toolbar, and click on the **Edit** link for the first Recipe content item in the list.

5. Towards the bottom of the node edit form, you will see inputs for our new compound field.

Summary

In this chapter, we began looking at ways to support HTML5 in Drupal 7, and integrated it with our `d7dev` site in a few different ways – with existing contributed modules and with code that we wrote ourselves. We learned about some of the differences between Microdata and RDFa, and did some extensive module development. We developed a custom compound field module, based on the Microdata specification at `http://schema.org/NutritionInformation`, allowing us to enhance our Recipe content type. However, at this point, our Nutrition Information Field module is still a bit rough around the edges. In the next chapter, we will introduce some more code examples, and clean up some of those rough edges.

Where to buy this book

You can buy Drupal 7 Development by Example Beginner's Guide from the Packt Publishing website: `http://www.packtpub.com/drupal-7-development-by-example-beginners-guide/book`.

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

www.PacktPub.com

For More Information:
www.packtpub.com/drupal-7-development-by-example-beginners-guide/book